Manual de Usuario de la clase TSimplex.

Proyecto: SimSEE
Archivo: mu_TSimplex.doc
Autor: Pablo Alfaro.
Fecha creación: 11/11/2009 16:13
Fecha última revisión: 11/11/2009 16:17
Institución: ADME - Montevideo - Uruguay.

Índice General

1) TSIMPLEX 3

1.1) Introducción. 3

1.2) Modo de Uso. 3

1.3) Árbol de herencias. 5

1.4) Propiedades. 6

1.5) Métodos Públicos 9

1) TSimplex

1.1) Introducción.

Esta clase permite modelar y resolver problemas de programación lineal del tipo

$$max_x c^T X$$
$$s.a.$$
$$Yx - c = 0$$
$$Ax - b \geq 0$$
$$X \in D, D \text{ un subconjunto de } R^n$$
$$X_i \text{ real}$$

Es modelada sobre una matriz con nf filas y nc columnas donde la fila nf es la función objetivo $c^T X$ y la columna nc son los términos independientes $\binom{-c}{-b}$. Por defecto todas las restricciones son consideradas de >= debiéndose indicar explícitamente si se desea declarar una restricción como de igualdad.
Se puede especificar además para cada variable en X cotas inferiores y superiores determinando el dominio D.

1.2) Modo de Uso.

Explicaremos la forma de utilizar la clase mediante el siguiente problema de ejemplo:

$$\min z = x1 + 3x2 + 2x3$$

$$s.a.$$
$$x1 + x2 + x3 \geq -10.5$$
$$x1 + x2 = -5.3$$
$$x1 - x3 \leq 2.9$$
$$0 \leq x1 \leq 12, -6 \leq x2 \leq 6, -5 \leq x3 \leq 5$$

Comenzamos llevando su función objetivo a una de maximización y sus restricciones a restricciones de = 0 o >= 0.

$$\max -z = -x1 - 3x2 - 2x3$$

$$s.a.$$

$$x1 + x2 + x3 + 10.5 \geq 0$$
$$x1 + x2 + 5.3 = 0$$
$$-x1 + x3 - 2.9 \geq 0$$
$$0 \leq x1 \leq 12, -6 \leq x2 \leq 6, -5 \leq x3 \leq 5$$

Teniendo el problema en esta forma estamos en condiciones de armar el simplex.

```
procedure ejemplo;
var
  i: Integer;
  spx: TSimplex;
begin
  //Creamos un simplex vacío cuya matriz M tendrá:
  //3 restricciones + la función objetivo
  //3 variables + los términos independientes
  spx:= TSimplex.Create_init(4, 4, NIL, NIL);

  //Cargamos la fila 1, pon_e(k, j, x) hace Mkj:= x
  spx.pon_e(1, 1, 1);
  spx.pon_e(1, 2, 1);
  spx.pon_e(1, 3, 1);
  spx.pon_e(1, spx.nc, 10.5);

  //Cargamos la fila 2 y la declaramos como de igualdad
  spx.pon_e(2, 1, 1);
  spx.pon_e(2, 2, 1);
  spx.pon_e(2, 3, 0);
  spx.pon_e(2, spx.nc, 5.3);
  spx.FijarRestriccionIgualdad(2);

  //Cargamos la fila 3
  spx.pon_e(3, 1, -1);
  spx.pon_e(3, 2, 0);
  spx.pon_e(3, 3, 1);
  spx.pon_e(3, spx.nc, -2.9);

  //Cargamos la fila objetivo z
  spx.pon_e(spx.nf, 1, -1);
  spx.pon_e(spx.nf, 2, -3);
  spx.pon_e(spx.nf, 3, -2);

  //cota_inf_set(i, x) fija la cota inferior de la variable en la
  //posición i a x, sota_sup_set hace lo propio con la cota superior
  //Cotas inferior y superior de x1
  spx.cota_inf_set(1, 0);
  spx.cota_sup_set(1, 12);

  //Cotas inferior y superior de x2
  spx.cota_inf_set(2, -6);
  spx.cota_sup_set(2, 6);

  //Cotas inferior y superior de x3
  spx.cota_inf_set(3, -5);
  spx.cota_sup_set(3, 5);

  //Vuelco el simplex al archivo 'ProblemaEjemplo.xlt' para verificar
  //que el problema armado sea el que quería
  spx.DumpSistemaToXLT('ProblemaEjemplo.xlt', '');

  //intento resolver
```

```
  if spx.resolver = 0 then
  begin
    //ok, encontró solución
    Writeln('Solución óptima encontrada:');
    //spx.fval obtiene el valor de z
    Writeln(#9'z= ', FloatToStrF(-spx.fval, ffGeneral, 8, 4));
    Writeln;
    for i:= 1 to 3 do
      //spx.xval(i) obtiene el valor de la variable i
      Writeln(#9, spx.fGetNombreVar(i), '= ', FloatToStrF(spx.xval(i),
                  ffGeneral, 8, 3));
    Writeln;
    for i:= 1 to 3 do
      //spx.yval(i) obtiene el valor de la restriccion i
      Writeln(#9, spx.fGetNombreRes(i), '= ', FloatToStrF(spx.yval(i),
                  ffGeneral, 8, 3));
    Writeln('Presione <Enter> para continuar');
    Readln;
  end
  else
    //Error, lanzamos la excepción
    raise Exception.Create('Error resolviendo simplex: ' +
                           spx.mensajeDeError);

  //Liberamos la memoria usada por el objeto
  spx.Free(true);
end;
```

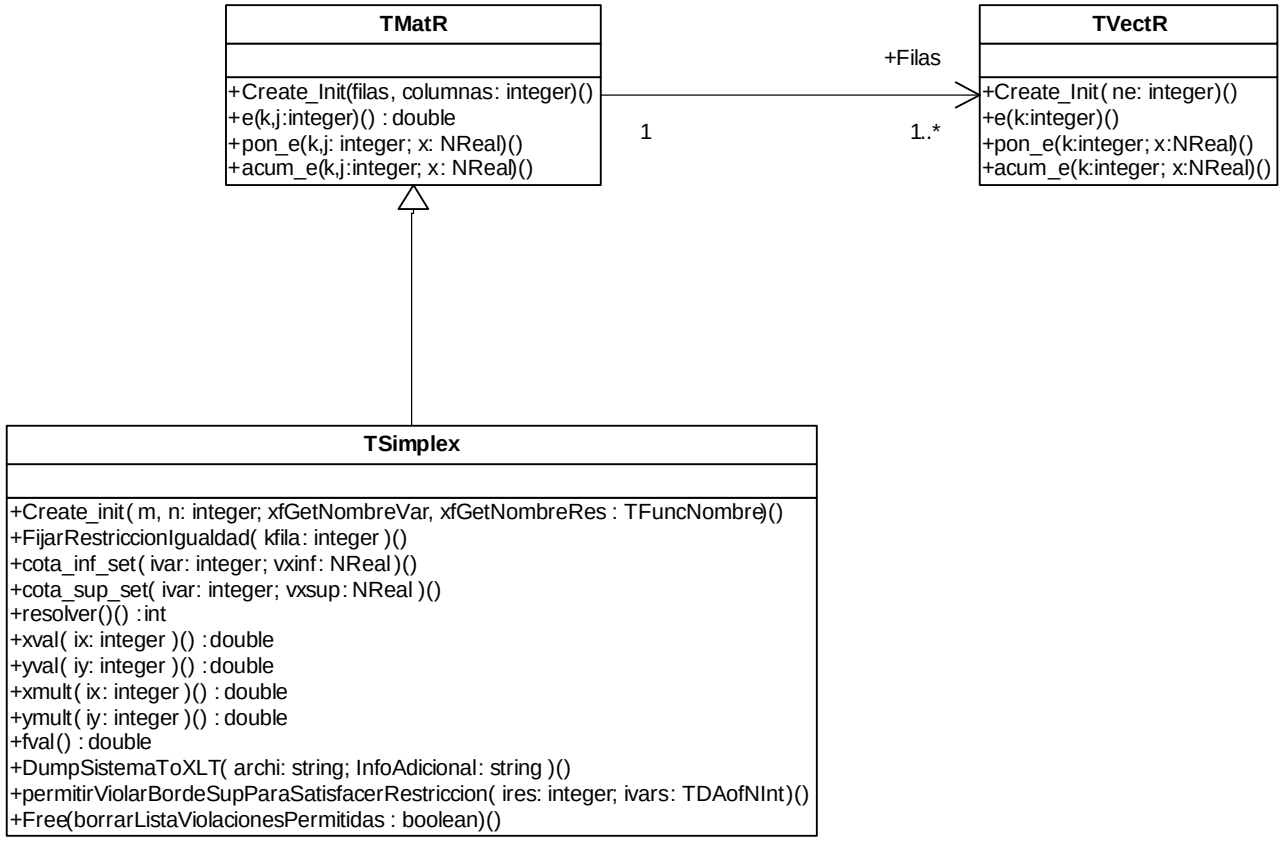
La salida en pantalla del programa será:

```
Solución Óptima encontrada:
      z= -10.1

      Var1= 0.7
      Var2= -6
      Var3= 3.6

      Res1= 8.8
      Res2= 0
      Res3= 0
Presione <Enter> para continuar
```

1.3) Árbol de herencias.



1.4) Propiedades.

```
      x_inf, x_sup: TVectR;
```

Contienen información sobre las cotas de las variables. Si la variable Xi esta acotada por $\hat{x}_i \leq x_i \leq \hat{x}_i$ => $x_{inf[i]} = \hat{x}_i , x_{sup[i]} = \hat{x}_i - \hat{x}_i$.

```
{$IFDEF GATILLOS_CAMBIOVAR}
      cnt_Gatillos: integer;
      gatillos_CambioVar: array of smallint;
      gatillos_no_procesados: boolean;
{$ENDIF}
```

//TODO

```
{$IFDEF CNT_COTAS}
      cnt_cotas_inf, cnt_cotas_sup: integer;
{$ENDIF}
```

Cuentan la cantidad de cotas inferiores y superiores que fueron fijadas mediante `cota_inf_set` y `cota_sup_set`.

```
      mensajeDeError: string;
```

Contiene en caso de haberlo un mensaje describiendo el error encontrado al llamar el método `resolver`

```
      violacionesPermitidas : TListaViolacionesPermitidasSimplex;
```

Contiene una lista de las violaciones a las cotas superiores que están permitidas en caso de no encontrar una solución factible al problema actual. Ver
`permitirViolarBordeSupParaSatisfacerRestriccion.`

```
fGetNombreVar, fGetNombreRes: TFuncNombre;
```

Funciones para obtener el nombre de la variable y la restricción en la posición i al momento de armar la matriz.

```
top, left: array of smallint;
```

abs(top[i]) guarda el número de fila original de la restricción o variable en la columna i.
abs(left[j]) guarda el número de fila original de la restricción o variable en la fila j.
Para determinar si se trata de una restricción o una variable se utiliza el signo, si su valor es negativo es una variable, si es positivo es una restricción.
Se indexan desde 1 y el primer elemento no se utiliza. Al iniciar a resolver top[i] = -i y left[j] = j.

Ejemplo
Si en la columna 8 se encuentra la restricción que se había creado en la fila 2 => top[8] = 2.
Si en la fila 1 se encuentra la variable que se había creado en la columna 3 => left[1] = -3.

```
iix: array of smallint; // > 0 están arriba, < 0 están abajo
iiy: array of smallint; // < 0 están arriba, > 0 están abajo
```

iix[i] guarda la posición que ocupa la variable que originalmente estaba en la columna i. Si iix[i] = -k la variable se encuentra en la fila k, si iix[i] = k la variable se encuentra en la columna k.

iiy[j] guarda la posición que ocupa la restricción que originalmente estaba en la fila j. Si iiy[j] = -m la variable se encuentra en la columna m, si iiy[j] = m la variable se encuentra en la fila m.
Se indexan desde 1 y el primer elemento no se utiliza. Al iniciar a resolver iix [i] = i y iiy [j] = j.

Ejemplo
Si la variable que originalmente estaba en la columna 8 se encuentra en la fila 3 => iix[8] = -3.
Si la restricción que originalmente estaba en la fila 1 se encuentra en la columna 4 => iiy [1] = -4.

```
cnt_RestrInfactibles: integer;
```

Guarda la cantidad de restricciones que aun no son factibles en la iteración actual.

```
cnt_igualdades: integer;
```

Guarda la cantidad de restricciones declaradas de igualdad mediante `FijarRestriccionIgualdad` (al inicio de A).

```
cnt_varfijas: integer;
```

Guarda la cantidad de variables fijadas mediante `FijarVariable`

```
cnt_columnasFijadas: Integer;
```

Guarda la cantidad de columnas fijadas con variables y restricciones

```
cnt_RestriccionesRedundantes: Integer;
```

Guarda la cantidad de restricciones declaradas como redundantes. Hasta esta fila inclusive las restricciones se consideran resueltas

```
flg_x: array of shortint; // es 0 si no hay cota superior,
```

Guardan información sobre las cotas de las variables
flg_x[i] = -1 si la variable actualmente en la columna i tiene cota superior, pero es la inferior la considerada actualmente
flg_x[i] = 1 si la variable actualmente en la columna i tiene cota superior y es la considerada actualmente
flg_x[i] = 2 si la variable actualmente en la columna i fue fijada
Se indexa desde 1

```
      flg_y: array of shortint;

Guarda información sobre los tipos de restricciones
```

flg_y[i] = 0 si la restricción actualmente en la fila i es de >= 0
flg_y[i] = 2 si la restricción actualmente en la fila i es de = 0

flg_y[i] = -2 si la restricción actualmente en la fila i es de = 0 pero se le cambio el signo

1.5) Métodos Públicos

Creación y reserva de memoria: **9**
Constructor Create_init(m, n: integer; xfGetNombreVar, xfGetNombreRes: TFuncNombre); virtual; 9

Armado y acceso a la matriz: **10**
function e(k,j:integer):NReal; virtual; 10
procedure pon_e(k,j: integer; x: NReal); virtual; 10
procedure acum_e(k,j:integer; x: NReal); virtual; 10
procedure FijarRestriccionIgualdad(kfila: integer); 10

Fijado de cotas para las variables de decisión: **11**
procedure cota_inf_set(ivar: integer; vxinf: NReal); 11
procedure cota_sup_set(ivar: integer; vxsup: NReal); 11

Resolución: **11**
function resolver: integer; virtual; 11

Acceso a resultados: **12**
function xval(ix: integer): NReal; virtual; 12
function yval(iy: integer): NReal; virtual; 12
function xmult(ix: integer): NReal; virtual; 12
function ymult(iy: integer): NReal; virtual; 13
function fval: NReal; virtual; 13

Depuración y funciones avanzadas: **14**
procedure DumpSistemaToXLT(archi: string; InfoAdicional: string); 14
procedure permitirViolarBordeSupParaSatisfacerRestriccion(ires: integer; ivars: TDAofNInt); 15

Destrucción y liberación de recursos: **16**
procedure Free(borrarListaViolacionesPermitidas : boolean); 16

Ejemplo de armado y resolución de un problema: **16**

Creación y reserva de memoria:

```
Constructor Create_init( m, n: integer; xfGetNombreVar,
                         xfGetNombreRes: TFuncNombre); virtual;
```

Crea un simplex con una matriz nula (todos sus elementos con el valor 0) de m filas por n columnas. Tener en cuenta que la función objetivo es una fila más y los términos independientes son una columna mas, por lo que si se quiere crear un simplex con 5 variables y 3 restricciones se debe pasar m= 4 y n= 6. Los parámetros xfGetNombreVar y xfGetNombreRes permiten especificar funciones de objeto para obtener el nombre de las variables o restricciones. Reciben un entero con el número de columna/fila que ocupa la variable/restricción al armar el problema.
Si se pasan con el valor NIL las variables tendrán por defecto el nombre “Vari” y las restricciones “Resj” donde i y j son la columna y fila que ocupan al armar el problema.

Ejemplo:

```
procedure crearSimplex();
var
      spx: TSimplex;
begin
      //Creamos un simplex con 3 restricciones y la función objetivo y
      //5 variables y los términos independientes.
      //Utilizamos los nombres por defecto
```

```
      spx:= TSimplex.Create_init(4, 6, NIL, NIL);
end;
```

Armado y acceso a la matriz:

```
function e(k,j:integer):NReal; virtual;
```

Obtiene el valor del elemento de la matriz en la fila k y columna j

Ejemplo:

```
var
      spx: TSimplex;
      m12: NReal;
begin
      …
      //Obtenemos el valor de la matriz en la fila 1 columna 2
      m12:= spx.e(1, 2);
      …
      …
end;

procedure pon_e(k,j: integer; x: NReal); virtual;
```

Fija en x el valor del elemento en la fila k y columna j de la matriz

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      //Fijamos el valor de la matriz en la fila 1 columna 2 a 10
      spx.pon_e(1, 2, 10);
      …
      …
end;

procedure acum_e(k,j:integer; x: NReal); virtual;
```

Suma x al valor del elemento en la fila k y columna j

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      //Sumamos 10 al valor de la matriz en la fila 1 columna 2
      spx.acum_e(1, 2, 10);
      …
      …
end;

procedure FijarRestriccionIgualdad( kfila: integer );
```

Declara a la restricción en la fila kfila como de igualdad. Por defecto todas las restricciones son de mayor o igual, debiéndose especificar explícitamente cuales se deben considerar de igualdad.

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      //Fijamos la restricción en la fila 5 como de igualdad
      spx.FijarRestriccionIgualdad(5);
      …
      …
end;
```

Fijado de cotas para las variables de decisión:

```
procedure cota_inf_set( ivar: integer; vxinf: NReal );
```

Fija la cota inferior de la variable ivar en vxinf

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      //Fijamos la cota inferior de la variable 1 en -2
      spx.cota_inf_set(1, -2);
      …
      …
end;
```

```
procedure cota_sup_set( ivar: integer; vxsup: NReal );
```

Fija la cota superior de la variable ivar en vxsup

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      //Fijamos la cota superior de la variable 1 en 2
      spx.cota_inf_set(1, 2);
      …
      …
end;
```

Resolución:

```
function resolver: integer; virtual;
```

```
Una vez armada la matriz con las ecuaciones y fijadas las cotas
superiores e inferiores de las variables correspondientes el método
resolver resuelve el simplex. Retorna 0 si encontró una solución
óptima factible, <> 0 si no. En caso de no encontrar solución la
variable mensajeDeError contiene una descripción del problema
encontrado.
```

Ejemplo:

```
var
      spx: TSimplex;
```

```
      z: NReal;
begin
      …
      //Armado del simplex
      …
      if spx.resolver = 0
            //Obtenemos el valor de la función objetivo tras resolver
            z:= spx.fval
      else
            raise Exception.Create('Error resolviendo simplex: ' +
                                    spx.mensajeDeError);
      …
      …
end;
```

Acceso a resultados:

```
function xval( ix: integer ): NReal; virtual;
```

Retorna el valor de una variable. ix es la columna que ocupaba la variable cuando se armó el simplex.

Ejemplo:

```
var
      spx: TSimplex;
      x1: NReal;
begin
      …
      //Obtenemos el valor de la variable que cargamos en la posición
      //1
      x1:= spx.xval(1);
      …
      …
end;
```

```
function yval( iy: integer ): NReal; virtual;
```

Retorna el valor de una restricción. iy es la fila que ocupaba la restricción cuando se armó el simplex.

Ejemplo:

```
var
      spx: TSimplex;
      y2: NReal;
begin
      …
      //Obtenemos el valor de la restricción que cargamos en la fila 2
      y2:= spx.yval(2);
      …
      …
end;
```

```
function xmult( ix: integer ): NReal; virtual;
```

Retorna el valor del multiplicador de Lagrange de una variable. ix es la columna que ocupaba la variable cuando se armó el simplex.

Ejemplo:

```
var
```

```
      spx: TSimplex;
      x1Mult: NReal;
begin
      …
      //Obtenemos el valor del multiplicador de Lagrange de la
      //variable que cargamos en la columna 1
      x1Mult:= spx.xmult(1);
      …
      …
end;

function ymult( iy: integer ): NReal; virtual;
```

Retorna el valor del multiplicador de Lagrange de una restricción. iy es la fila que ocupaba la restricción cuando se armó el simplex.

Ejemplo:

```
var
      spx: TSimplex;
      y2Mult: NReal;
begin
      …
      //Obtenemos el valor del multiplicador de Lagrange de la
      //restricción que cargamos en la fila 2
      y1Mult:= spx.ymult(2);
      …
      …
end;

function fval: NReal; virtual;
```

Retorna el valor obtenido de la función objetivo (fila nf de la matriz) al resolver el simplex. Tener en cuenta que el problema se debe escribir como problema de maximización, por lo que si nuestro problema original era de minimización debemos considerar –spx.fval

Ejemplo:

```
var
      spx: TSimplex;
      z: NReal;
begin
      …
      Armado del simplex
      …
      if spx.resolver = 0
            //Obtenemos el valor de la función objetivo tras resolver
            z:= spx.fval
      else
            raise Exception.Create('Error resolviendo simplex: ' +
                                    spx.mensajeDeError);
      …
      …
end;
```

Depuración y funciones avanzadas:

```
procedure DumpSistemaToXLT(archi: string; InfoAdicional: string);
                           overload; virtual;
```

Vuelca los datos contenidos en el simplex a un archivo. Es útil para verificar que una vez que se armó el problema este sea el problema deseado.
Al definir las cotas de una variable el simplex realiza un cambio de variable con respecto a la original donde si xi esta acotada por $\breve{x}i \leq xi \leq \hat{x}i$, se representa como $xi^* = xi - \breve{x}i$ con $0 \leq xi^* \leq \hat{x}i - \breve{x}i$. Es importante tener esto en cuenta a la hora de usar el método `DumpSistemaToXLT` ya que este imprime $\breve{x}i$ en la fila de las x_inf, pero $\hat{x}i - \breve{x}i$ en la fila de las x_sup.

El formato de salida es el siguiente:

InfoAdicional:
Información adicional que se haya incluido al llamar a `DumpSistemaToXLT`

NEnteras: nroEnteras [iEntera1] [iEntera2] …. [iEnteraNroEnteras]
nroEnteras es el número de variables enteras en el simplex
Los valores iEnterak indican el índice de la columna original de la variable entera número k

ivae VarAcoplada ResAcoplada
Si hay variables enteras con acoples indican para cada ivae, índice de variable entera, (de 0 a nroEnteras-1), la variable y restricción acopladas indicadas por su índice de fila o columna al armar el simplex

cnt_varfijas:
Número de variables cuyo valor haya sido fijado
cnt_RestriccionesRedundantes:
Número de restricciones que hayan sido declaradas redundantes

cnt_violacionesUsadas
Número de violaciones permitidas utilizadas (ver `permitirViolarBordeSupParaSatisfacerRestriccion`)

violacionesPermitidas
violacionesPermitidas.Count=
Cantidad fichas de violaciones permitidas
ires usada iViolacionAUsar nIvars ivars[]
ires es el índice de la fila original de la restricción
usada indica si se uso o no la violación
iViolacionAUsar es el índice de la próxima variable a modificar la cota superior para resolver una restricción
nIvars es el tamaño del arreglo con las variables cuya cota superior es violable
ivars es el arreglo con las variables cuya cota superior es violable
****

x: nombre de la variable
x_inf: cota inferior $\breve{x}i$ de la variable
x_sup: cota superior menos cota inferior de la variable $\hat{x}i - \breve{x}i$
flg_x: banderas de las variables, dan información sobre las cotas de la variable
es 0 si la variable no tiene cota superior
es -1 si la variable tiene cota superior, pero es la inferior la considerada actualmente
es 1 si la variable tiene cota superior y es la considerada actualmente en el sistema
es 2 si la variable fue fijada.
flg_y: banderas de las restricciones, dan información sobre el tipo de restricción
0 para restricciones de >=0
2 si es una =0
-2 si es una =0 a la que le cambié el signo
top: Guardan el índice original k (cuando fue armada la matriz) de la variable que está ahora en

left: la posición i. Para las variables se guarda –k y para las restricciones k
Si la variable que está en la columna 8 había comenzado en la columna => top[8] = -2. El arreglo left contiene lo propio para las filas. Se indexan desde 1

----
sistema ----------
.....

A partir de aquí se muestra la matriz del simplex tal como la contiene el objeto. Al final de cada fila se muestra que condición debe cumplir el valor actual de la restricción (ti, término independiente) para considerarse satisfecha. En caso de que la fila este ocupada por una variable con cotas superior e inferior se muestran ambas condiciones.

Ejemplo:

```
var
      spx: TSimplex;
      z: NReal;
begin
      …
      Armado del simplex
      …
      spx.DumpSistemaToXLT('ProblemaArmado.xlt', '');
      //Verifico que el problema en el archivo sea el que realmente
      //deseo
      if spx.resolver = 0
            //Obtenemos el valor de la función objetivo tras resolver
            z:= spx.fval
      else
            raise Exception.Create('Error resolviendo simplex: ' +
                                    spx.mensajeDeError);
      …
      …
end;

procedure permitirViolarBordeSupParaSatisfacerRestriccion( ires:
                  integer; ivars: TDAofNInt );
```

Declara que las variables en ivars pueden agrandar su cota superior si la restricción ires no puede satisfacerse de otra manera.
Debe ser llamado con los índices que tenían la restricción y las variables cuando se cargaron en el Simplex.
Esto se hizo para poder resolver las restricciones de cota superior de volumen almacenado. Si un embalse esta cerca de estar lleno y tiene un volumen de aportes importante puede suceder que el vertimiento máximo no nos permita vaciar lo suficiente el embalse. Para solucionar esta situación se permite especificar que la cota de la variable "Volumen Vertido" del embalse puede violarse en caso de no poder resolver el problema por la restricción de volumen máximo almacenado del embalse con la cota fijada. Para cada restricción que se desee poder resolver de esta forma debe especificarse su número de fila y el número de columna de las variables a las que se les pueda violar la cota superior. Estos números refieren a las posiciones que ocupaban la restricción y las variables al armar el simplex.

Destrucción y liberación de recursos:

```
procedure Free(borrarListaViolacionesPermitidas : boolean);
```

Destruye el objeto simplex y libera la memoria que tuviera asignada. El parámetro borrarListaViolacionesPermitidas responde a una necesidad de implementación de la clase TMIPSimplex, usando esta clase se debe pasar siempre con true.

Ejemplo:

```
var
      spx: TSimplex;
begin
      …
      spx.Free(true);
      …
      …
end;
```